Complexity of Propositional Logics in Team Semantics

Miika Hannula[1], Juha Kontinen[1], Jonni Virtema[2], and Heribert Vollmer[2]

[1] University of Helsinki, Department of Mathematics and Statistics, Helsinki, Finland {miika.hannula,juha.kontinen}@helsinki.fi
[2] Leibniz Universität Hannover, Institut für Theoretische Informatik, Germany jonni.virtema@gmail.com, vollmer@thi.uni-hannover.de

Abstract. We classify the computational complexity of the satisfiability, validity and model-checking problems for propositional independence, inclusion, and team logic. Our main result shows that the satisfiability and validity problems for propositional team logic are complete for alternating exponential-time with polynomially many alternations.

Keywords: Propositional logic, team semantics, dependence, independence, inclusion, satisfiability, validity, model-checking

1 Introduction

Dependence logic [22] is a new logical framework for formalising and studying various notions of dependence and independence that are important in many scientific disciplines such as experimental physics, social choice theory, computer science, and cryptography. Dependence logic extends first-order logic by dependence atoms

$$\mathrm{dep}(x_1, \ldots, x_n, y) \tag{1}$$

expressing that the value of the variable y is functionally determined on the values of $x_1, \ldots, x_n$. Satisfaction for formulas of dependence logic is defined using sets of assignments (*teams*) and not in terms of single assignments as in first-order logic. Whereas dependence logic studies the notion of functional dependence, independence and inclusion logic (introduced in [8] and [7], respectively) formalize the concepts of independence and inclusion. Independence logic (inclusion logic) is obtained from dependence logic by replacing dependence atoms by the so-called independence atoms $\boldsymbol{x} \perp_{\boldsymbol{y}} \boldsymbol{z}$ (inclusion atoms $\boldsymbol{x} \subseteq \boldsymbol{y}$). The intuitive meaning of the independence atom is that the variables of the tuples $\boldsymbol{x}$ and $\boldsymbol{z}$ are independent of each other for any fixed value of the variables in $\boldsymbol{y}$, whereas the inclusion atom declares that all values of the tuple $\boldsymbol{x}$ appear also as values of $\boldsymbol{y}$. In database theory these atoms correspond to the so-called embedded multivalued dependencies and inclusion dependencies (see, e.g., [9]). Independence atoms have also a close connection to conditional independence in statistics.

The topic of this article is propositional team semantics which has received relatively little attention so far. On the other hand, modal team semantics has

Table 1: Overview of the results (completeness results if not stated otherwise)

	SAT	VAL	MC
$\mathrm{PL}[\perp_{\mathrm{c}}]$	NP	in $\mathsf{coNEXPTIME}^{\mathsf{NP}}$	NP
$\mathrm{PL}[\subseteq]$	EXPTIME [12]	coNP	P [11]
$\mathrm{PL}[\sim], \mathrm{PL}[\perp_{\mathrm{c}}, \subseteq, \sim]$	AEXPTIME(poly)	AEXPTIME(poly)	PSPACE [19]

been studied actively. Since the propositional logics studied in the article are fragments of the corresponding modal logics, some upper bounds trivally transfer to the propositional setting. The study of propositional team semantics as a subject of independent interest was initiated after surprising connections between propositional team semantics and the so-called *inquisitive semantics* was discovered (see [24] for details). The first systematic study on the expressive power of propositional dependence logic and many of its variants is due to [24,25]. In the same works natural deduction type inference systems for these logics are also developed, whereas in [21] a complete Hilbert-style axiomatization and a labeled tableaux calculus for propositional dependence logic is presented. Very recently Hilbert-style proof systems for related logics that incorporate the classical negation have been introduced by Lück, see [17].

The computational aspects of (first-order) dependence logic and its variants have been actively studied, and are now quite well understood (see [5]). On the other hand, the complexity of the propositional versions of these logics have not been systematically studied. The study was initiated in [23] where the validity problem of propositional dependence logic was shown to be NEXPTIME-complete. Also recently propositional inclusion logic has been studied in the article [12] and in the manuscript [11]. In this article we study the complexity of satisfiability, validity and model-checking of propositional independence, inclusion and team logic that extends propositional logic by the classical negation. The classical negation has turned out to be a very powerful connective in the settings of first-order and modal team semantics, see e.g., [13] and [14]. Our results (see Table 1) show that the same is true in the propositional setting. In particular, our main result shows that the satisfiability and validity problems of team logic are complete for alternating exponential time with polynomially many alternations (AEXPTIME(poly)). The results hold also for the extensions of propositional inclusion and independence logic by the classical negation. Recently levels of the exponential hierarchy have been logically characterized in the context of propositional team semantics, in [18,10].

2 Preliminaries

In this section we define the basic concepts and results relevant to team-based propositional logics. We assume that the reader is familiar with propositional logic.

2.1 Syntax and semantics

Let D be a finite, possibly empty, set of proposition symbols. A function $s : D \to \{0,1\}$ is called an *assignment*. A set X of assignments $s : D \to \{0,1\}$ is called a *team*. The set D is the *domain* of X. We denote by 2^D the set of *all assignments* $s : D \to \{0,1\}$.

Let Φ be a set of proposition symbols. The syntax for propositional logic $\mathrm{PL}(\Phi)$ is defined as follows.

$$\varphi ::= p \mid \neg p \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi), \qquad \text{where } p \in \Phi.$$

We write $\mathrm{Var}(\varphi)$ for the set of all proposition symbols that appear in φ. We denote by $\models_{\mathrm{PL}}$ the ordinary satisfaction relation of propositional logic defined via assignments in the standard way. Next we give team semantics for propositional logic.

Definition 1. *Let Φ be a set of proposition symbols and let X be a team. The satisfaction relation $X \models \varphi$ is defined as follows.*

$$\begin{aligned}
X \models p \quad &\Leftrightarrow \quad \forall s \in X : s(p) = 1.\\
X \models \neg p \quad &\Leftrightarrow \quad \forall s \in X : s(p) = 0.\\
X \models (\varphi \wedge \psi) \quad &\Leftrightarrow \quad X \models \varphi \text{ and } X \models \psi.\\
X \models (\varphi \vee \psi) \quad &\Leftrightarrow \quad Y \models \varphi \text{ and } Z \models \psi, \text{ for some } Y, Z \text{ such that } Y \cup Z = X.
\end{aligned}$$

Note that in team semantics $\neg$ is not the classical negation (denoted by $\sim$ in this article) but a so-called *dual* negation that does not satisfy the law of excluded middle. Next proposition shows that the team semantics and the ordinary semantics for propositional logic defined via assignments coincide.

Proposition 1 ([22]). *Let φ be a formula of propositional logic and let X be a propositional team. Then $X \models \varphi$ iff $\forall s \in X : s \models_{PL} \varphi$.*

The syntax of *propositional dependence logic* $\mathrm{PD}(\Phi)$ is obtained by extending the syntax of $\mathrm{PL}(\Phi)$ by the rule

$$\varphi ::= \mathrm{dep}(p_1, \ldots, p_n, q)\,, \qquad \text{where } p_1, \ldots, p_n, q \in \Phi.$$

The semantics for the propositional dependence atoms are defined as follows:

$$\begin{aligned}
X \models \mathrm{dep}(p_1, \ldots, p_n, q) \quad \Leftrightarrow \quad &\forall s, t \in X : s(p_1) = t(p_1), \ldots, s(p_n) = t(p_n)\\
&\text{implies that } s(q) = t(q).
\end{aligned}$$

The next proposition is very useful when determining the complexity of PD, and it is proved analogously as for first-order dependence logic [22].

Proposition 2 (Downwards closure). *Let φ be a PD-formula and let $Y \subseteq X$ be propositional teams. Then $X \models \varphi$ implies $Y \models \varphi$.*

In this article we study the variants of PD obtained by replacing dependence atoms in terms of the so-called independence or inclusion atoms: The syntax of *propositional independence logic* $\mathrm{PL}[\perp_{\mathrm{c}}](\Phi)$ is obtained by extending the syntax of $\mathrm{PL}(\Phi)$ by the rule

$$\varphi ::= \boldsymbol{q} \perp_{\boldsymbol{p}} \boldsymbol{r},$$

where $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ are finite tuples of proposition symbols (not necessarily of the same length). The syntax of *propositional inclusion logic* $\mathrm{PL}[\subseteq](\Phi)$ is obtained by extending the syntax of $\mathrm{PL}(\Phi)$ by the rule

$$\varphi ::= \boldsymbol{p} \subseteq \boldsymbol{q},$$

where $\boldsymbol{p}$ and $\boldsymbol{q}$ are finite tuples of proposition symbols with the same length. Satisfaction for these atoms is defined as follows. If $\boldsymbol{p} = (p_1, \dots, p_n)$ and s is an assignment, we write $s(\boldsymbol{p})$ for $(s(p_1), \dots, s(p_n))$.

$$\begin{aligned} X \models \boldsymbol{q} \perp_{\boldsymbol{p}} \boldsymbol{r} \quad &\Leftrightarrow \quad \forall s, t \in X: \text{ if } s(\boldsymbol{p}) = t(\boldsymbol{p}) \\ &\qquad \text{then there exists } u \in X : u(\boldsymbol{pq}) = s(\boldsymbol{pq}) \text{ and } u(\boldsymbol{r}) = t(\boldsymbol{r}). \\ X \models \boldsymbol{p} \subseteq \boldsymbol{q} \quad &\Leftrightarrow \quad \forall s \in X \exists t \in X : s(\boldsymbol{p}) = t(\boldsymbol{q}). \end{aligned}$$

It is easy to check that neither $\mathrm{PL}[\perp_{\mathrm{c}}]$ nor $\mathrm{PL}[\subseteq]$ is a downward closed logic (cf. Proposition 2). However, analogously to first-order inclusion logic [7], the formulas of $\mathrm{PL}[\subseteq]$ have the following closure property.

Proposition 3 (Closure under unions). *Let $\varphi \in \mathrm{PL}[\subseteq]$ and let X_i, for $i \in I$, be teams. Suppose that $X_i \models \varphi$, for each $i \in I$. Then $\bigcup_{i \in I} X_i \models \varphi$.*

We will also consider the extensions of PL, $\mathrm{PL}[\perp_{\mathrm{c}}]$ and $\mathrm{PL}[\subseteq]$, by the classical negation $\sim$ with the standard semantics:

$$X \models {\sim}\varphi \Leftrightarrow X \not\models \varphi.$$

These extensions are denoted by $\mathrm{PL}[\sim]$ (propositional team logic), $\mathrm{PL}[\perp_{\mathrm{c}}, \sim]$ and $\mathrm{PL}[\subseteq, \sim]$, respectively.

A general notion of a *generalized dependency atom* expressing a property of a propositional team has also been studied in the literature. For the purposes of this article precise definitions are not required and are thus omitted, for a detailed exposition for generalised dependency atoms see, e.g., [10]. We say that a generalized dependency atom A has a polynomial time checkable semantics if $X \models A(\boldsymbol{p})$ can be decided in polynomial time with respect to the combined size of X and $\boldsymbol{p}$. Each of the atoms defined above are examples of generalized dependency atoms. It is easy to see that each of these atoms has a polynomial time checkable semantics.

2.2 Auxiliary operators

The following additional operators will be used in this paper:

$$
\begin{aligned}
X \models \varphi \varovee \psi \quad &\Leftrightarrow \quad X \models \varphi \text{ or } X \models \psi, \\
X \models \varphi \otimes \psi \quad &\Leftrightarrow \quad \forall Y, Z \subseteq X: \text{ if } Y \cup Z = X, \text{ then } Y \models \varphi \text{ or } Z \models \psi, \\
X \models \varphi \multimap \psi \quad &\Leftrightarrow \quad \forall Y \subseteq X: \text{if } Y \models \varphi, \text{ then } Y \models \psi, \\
X \models \max(x_1, \dots, x_n) \quad &\Leftrightarrow \quad \{(s(x_1), \dots, s(x_n)) \mid s \in X\} = \{0,1\}^n.
\end{aligned}
$$

If $X \models \max(\boldsymbol{x})$, we say that X is *maximal over* $\boldsymbol{x}$. If tuples $\boldsymbol{x}$ and $\boldsymbol{y}$ are pairwise disjoint and $X \models \max(\boldsymbol{x}) \wedge \boldsymbol{x} \perp \boldsymbol{y}$, then we say that X is *maximal over* $\boldsymbol{x}$ *for all* $\boldsymbol{y}$.

Proposition 4. *The operators* $\mathrm{dep}(\cdot)$, $\varovee$, $\otimes$, $\multimap$, *and* $\max(\cdot)$ *have uniform polynomial size translations in* $\mathrm{PL}[\sim]$.

Proof. We present the following translations of which item 3 is due to [19] and item 4 uses the idea of [1].

1. The connective $\otimes$ is actually the dual of $\vee$, and hence $\varphi \otimes \psi$ can be written as $\sim(\sim\varphi \vee \sim\psi)$.
2. Intuitionistic disjunction $\varphi \varovee \psi$ can be written as $\sim(\sim\varphi \wedge \sim\psi)$.
3. Intuitionistic implication $\varphi \multimap \psi$ can be expressed as $(\sim\varphi \varovee \psi) \otimes \sim(p \vee \neg p)$.
4. First note that $\mathrm{dep}(x)$ can be written as $x \varovee \neg x$. Using this we can write $\mathrm{dep}(x_1, \dots, x_n, y)$ as $\bigwedge_{i=1}^{n} \mathrm{dep}(x_i) \multimap \mathrm{dep}(y)$.
5. We show that $\max(x_1, \dots, x_n)$ is equivalent to $\sim \bigvee_{i=1}^{n} \mathrm{dep}(x_i)$. Assume first that $X \models \bigvee_{i=1}^{n} \mathrm{dep}(x_i)$, we show that $X \not\models \max(x_1, \dots, x_n)$. By the assumption, we find $Y_1, \dots, Y_n \in X$, $\bigcup_{i=1}^{n} Y_i = X$, such that $Y_i \models =\!(x_i)$. Now for all i there exists a $b_i \in \{0,1\}$ such that if $Y_i \neq \emptyset$, then for all $s \in Y_i$, $s(x_i) \neq b_i$. Since the assignment $x_i \mapsto b_i$ is not in X, we obtain that $X \not\models \max(x_1, \dots, x_n)$.
   Assume then that $X \not\models \max(x_1, \dots, x_n)$, we show that $X \models \bigvee_{i=1}^{n} \mathrm{dep}(x_i)$. By the assumption there exists a boolean sequence $(b_1, \dots, b_n)$ such that for no $s \in X$ we have $s(x_i) = b_i$ for all $i = 1, \dots, n$. Let $Y_i := \{s \in X \mid s(x_i) \neq b_i\}$. Since then $X = \bigcup_{i=1}^{n} Y_i$ and $Y_i \models =\!(x_i)$, we obtain that $X \models \bigvee_{i=1}^{n} \mathrm{dep}(x_i)$.

□

2.3 Satisfiability, validity, and model checking in team semantics

Next we define satisfiability and validity in the context of team semantics. Let L be a logic with team semantics. A formula $\varphi \in L$ is *satisfiable*, if there exists a non-empty team X such that $X \models \varphi$. A formula $\varphi \in L$ is *valid*, if $X \models \varphi$ holds for every non-empty team X such that the proposition symbols that occur in φ are in the domain of X.[3] Note that when the team is empty, satisfaction becomes easy to decide, see Proposition 5 below.

[3] It is easy to show that all of the logics considered in this article have the so-called locality property, i.e., satisfaction of a formula depends only on the values of the proposition symbols that occur in the formula [7].

Table 2: Complexity of satisfiability, validity, and model checking of PL and PD. All results are completeness results.

	SAT	VAL	MC	References
PL	NP	coNP	NC^1	[4,15,2]
PD	NP	NEXPTIME	NP	[16,6,23]

The satisfiability problem $\mathsf{SAT}(L)$ and the validity problem $\mathsf{VAL}(L)$ are then defined in the obvious manner: Given a formula $\varphi \in L$, decide whether the formula is satisfiable (valid, respectively). The variant of the model checking problem that we are concerned with in this article is the following: Given a formula $\varphi \in L$ and a team X, decide whether $X \models \varphi$. See Table 2 for known complexity results on PL and PD.

Proposition 5. *Checking whether* $\emptyset \models \varphi$*, for* $\varphi \in \mathrm{PL}[\perp_{\mathrm{c}} \subseteq, \sim]$*, can be done in* P*. Furthermore,* $\emptyset \models \varphi$ *for all* $\varphi \in \mathrm{PL}[\perp_{\mathrm{c}} \subseteq]$.

Proof. Define a function $\pi : \mathrm{PL}[\perp_{\mathrm{c}}, \subseteq, \sim] \to \{0,1\}$ recursively as follows. Note that addition is mod 2.

- If $\varphi \in \{p, \neg p, \boldsymbol{q} \perp_{\boldsymbol{p}} \boldsymbol{r}, \boldsymbol{p} \subseteq \boldsymbol{q}\}$, then $\pi(\varphi) = 1$.
- If $\varphi = \psi_0 \wedge \psi_1$, then $\pi(\varphi) = \pi(\psi_0) \cdot \pi(\psi_1)$.
- If $\varphi = \psi_0 \vee \psi_1$, then $\pi(\varphi) = \pi(\psi_0) \cdot \pi(\psi_1)$.
- If $\varphi = \sim\psi$, then $\pi(\varphi) = \pi(\psi) + 1$.

It is easy to check that $\emptyset \models \varphi$ iff $\pi(\varphi) = 1$. Since $\pi(\varphi)$ can be computed in P, the claim follows.

3 Complexity of Satisfiability and Validity

In this section we consider the complexity of the satisfiability and validity problems for propositional independence, inclusion and team logic.

3.1 The logics PL[$\perp_{\mathrm{c}}$] and PL[$\subseteq$]

We consider first the complexity of $\mathsf{SAT}(\mathrm{PL}[\perp_{\mathrm{c}}])$. The following simple lemma turns out to be very useful.

Lemma 1. *Let* $\varphi \in \mathrm{PL}[\perp_{\mathrm{c}}]$ *and* X *a team such that* $X \models \varphi$*. Then* $\{s\} \models \varphi$*, for all* $s \in X$*.*

Proof. The claim is proved using induction on the construction of φ. It is easy to check that a singleton team satisfies all independence atoms, and the cases corresponding to disjunction and conjunction are straightforward.

Theorem 1. $\mathsf{SAT}(\mathrm{PL}[\perp_c])$ *is complete for* NP.

Proof. Note first that since $\mathsf{SAT}(\mathrm{PL})$ is NP-complete, it follows by Proposition 1 that $\mathsf{SAT}(\mathrm{PL}[\perp_c])$ is NP-hard. For containment in NP, note that by Lemma 1, a formula $\varphi \in \mathrm{PL}[\perp_c]$ is satisfiable iff it is satisfied by some singleton team $\{s\}$. It is immediate that for any s, $\{s\} \models \varphi$ iff $\{s\} \models \varphi^T$, where $\varphi^T \in \mathrm{PL}$ is acquired from φ by replacing all independence atoms by $(p \vee \neg p)$. Thus it follows that φ is satisfiable iff φ^T is satisfiable. Therefore, the claim follows. □

Next we consider the complexity of $\mathsf{VAL}(\mathrm{PL}[\perp_c])$.

Theorem 2. $\mathsf{VAL}(\mathrm{PL}[\perp_c])$ *is hard for* $\mathsf{NEXPTIME}$ *and is in* $\mathsf{coNEXPTIME}^{\mathsf{NP}}$.

Proof. Since the dependence atom $\mathrm{dep}(\boldsymbol{x}, y)$ is equivalent to the independence atom $y \perp_{\boldsymbol{x}} y$ and $\mathsf{VAL}(\mathrm{PD})$ is $\mathsf{NEXPTIME}$-complete [23], hardness for $\mathsf{NEXPTIME}$ follows. We will show in Theorem 9 on p. 13 that the model checking problem for $\mathrm{PL}[\perp_c]$ is complete for NP. It then follows that the complement of the problem $\mathsf{VAL}(\mathrm{PL}[\perp_c])$ is in $\mathsf{NEXPTIME}^{\mathsf{NP}}$: the question whether φ is in the complement of $\mathsf{VAL}(\mathrm{PL}[\perp_c])$ can be decided by guessing a subset X of 2^D, where D contains the set of proposition symbols appearing in φ, and checking whether $X \not\models \varphi$. Therefore $\mathsf{VAL}(\mathrm{PL}[\perp_c]) \in \mathsf{coNEXPTIME}^{\mathsf{NP}}$. □

Next we turn to propositional inclusion logic.

Theorem 3 ([12]). $\mathsf{SAT}(\mathrm{PL}[\subseteq])$ *is complete for* $\mathsf{EXPTIME}$.

We end this section by determining the complexity of $\mathsf{VAL}(\mathrm{PL}[\subseteq])$.

Theorem 4. $\mathsf{VAL}(\mathrm{PL}[\subseteq])$ *is complete for* coNP.

Proof. Recall that PL is a sub-logic of $\mathrm{PL}[\subseteq]$, and hence $\mathsf{VAL}(\mathrm{PL}[\subseteq])$ is hard for coNP. Therefore, it suffices to show $\mathsf{VAL}(\mathrm{PL}[\subseteq]) \in \mathsf{coNP}$. It is easy to check that, by Proposition 3, a formula $\varphi \in \mathrm{PL}[\subseteq]$ is valid iff it is satisfied by all singleton teams $\{s\}$. Note also that, over a singleton team $\{s\}$, an inclusion atom $(p_1, \ldots, p_n) \subseteq (q_1, \ldots, q_n)$ is equivalent to the PL-formula

$$\bigwedge_{1 \le i \le n} p_i \leftrightarrow q_i.$$

Denote by φ^* the PL-formula acquired by replacing all inclusion atoms in φ by their PL-translations. By the above, φ is valid iff φ^* is valid. Since $\mathsf{VAL}(\mathrm{PL})$ is in coNP the claim follows. □

3.2 Logics with the classical negation

Next we incorporate classical negation in our logics. The main result of this section shows that the satisfiability and validity problems for $\mathrm{PL}[\sim]$ are complete for $\mathsf{AEXPTIME(poly)}$. The result holds also for $\mathrm{PL}[\mathcal{C}, \sim]$ where $\mathcal{C}$ is any finite collection of dependency atoms with polynomial-time checkable semantics. This

covers the standard dependency notions considered in the team semantics literature. The upper bound follows by an exponential-time alternating algorithm where alternation is bounded by formula depth. For the lower bound we first relate $\mathsf{AEXPTIME}(\mathsf{poly})$ to polynomial-time alternating Turing machines that query to oracles obtained from a quantifier prefix of polynomial length. We then show how to simulate such computations in $\mathrm{PL}[\sim]$.

First we observe that the classical negation gives rise to polynomial-time reductions between the validity and the satisfiability problems. Hence, we restrict our attention to satisfiability hereafter.

Proposition 6. *Let $\varphi \in \mathrm{PL}[\mathcal{C}, \sim]$ where $\mathcal{C} \subseteq \{\mathrm{dep}(\cdot), \bot_{\mathrm{c}}, \subseteq\}$. Then one can construct in polynomial time formulae $\psi, \theta \in \mathrm{PL}[\mathcal{C}, \sim]$ such that*

(i) φ is satisfiable iff ψ is valid, and
(ii) φ is valid iff θ is satisfiable.

Proof. We define

$$\psi := \max(\boldsymbol{x}) \multimap ((p \vee \neg p) \vee (\varphi \wedge \sim(p \wedge \neg p))),$$
$$\theta := \max(\boldsymbol{x}) \wedge (\sim(p \wedge \neg p) \multimap \varphi),$$

where $\boldsymbol{x}$ lists $\mathrm{Var}(\varphi)$. Note that $X \models \sim(p \wedge \neg p)$ iff X is non-empty. It is straightforward to show that (*i*) and (*ii*) hold. Also by Proposition 4, ψ and θ can be constructed in polynomial time from φ. ☐

Next we show the upper bound for the satisfiability problem of propositional logic with the classical negation, and the independence and inclusion atoms.

Theorem 5. $\mathsf{SAT}(\mathrm{PL}[\bot_{\mathrm{c}}, \subseteq, \sim]) \in \mathsf{AEXPTIME}(\mathsf{poly})$.

Proof. Let $\varphi \in \mathrm{PL}[\bot_{\mathrm{c}}, \subseteq, \sim]$. First existentially guess a possibly exponential-size team T with domain $\mathrm{Var}(\varphi)$. Then implement Algorithm 1 (see Appendix) on $\textsc{mc}(T, \varphi, 1)$. The result follows since this algorithm runs in polynomial time and its alternation is bounded by the size of φ. ☐

Let us then turn to the lower bound. We show that the satisfiability problem of $\mathrm{PL}[\sim]$ is hard for $\mathsf{AEXPTIME}(\mathsf{poly})$. For this, we first relate $\mathsf{AEXPTIME}(\mathsf{poly})$ to oracle quantification for polynomial-time oracle Turing machines. This approach is originally due to Orponen in [20], where the classes Σ_k^{EXP} and Π_k^{EXP} of the exponential-time hierarchy were characterized. Recall that the exponential-time hierarchy corresponds to the class of problems that can be recognized by an exponential-time alternating Turing machine with constantly many alternations. In the next theorem we generalize Orponen's characterization to exponential-time alternating Turing machines with *polynomially* many alternations (i.e. the class $\mathsf{AEXPTIME}(\mathsf{poly})$) by allowing quantification of polynomially many oracles.

By $(A_1, \ldots, A_k)$ we denote an efficient disjoint union of sets $A_1, \ldots, A_k$, e.g., $(A_1, \ldots, A_k) = \{(i, x) : x \in A_i, 1 \leq i \leq k\}$.

Theorem 6. *A set A belongs to the class* $\mathsf{AEXPTIME(poly)}$ *iff there exist a polynomial f and a polynomial-time alternating oracle Turing machine M such that, for all x,*

$$x \in A \textit{ iff } Q_1 A_1 \ldots Q_{f(n)} A_{f(n)} (M \textit{ accepts } x \textit{ with oracles } (A_1, \ldots, A_{f(n)})),$$

where n is the length of x and $Q_1, \ldots, Q_{f(n)}$ alternate between $\exists$ and $\forall$, i.e., $Q_{i+1} \in \{\forall, \exists\} \setminus \{Q_i\}$.

Proof. The proof is a straightforward generalization of the proof of Theorem 5.2. in [20]:

If-part. Let M be a polynomial-time alternating oracle Turing machine, and let f and p be polynomials that bound the length of the oracle quantification and the running time of M, respectively. We describe the behaviour of an alternating Turing machine M' such that for all x,

$$M' \text{ accepts } x \text{ iff } Q_1 A_1 \ldots Q_{f(n)} A_{f(n)} (M \text{ accepts } x \text{ with oracle } (A_1, \ldots, A_{f(n)})). \tag{2}$$

At first, M' simulates the quantifier block $Q_1 A_1 \ldots Q_{f(n)} A_{f(n)}$ in $f(n)$ consecutive steps. Namely, for $1 \leq k \leq f(n)$ where $Q_k = \exists$ (or $Q_k = \forall$), M' **existentially** (**universally**) chooses a set A_k that consists of strings i of lenght at most $p(n)$. Then M' evaluates the computation tree associated with the Turing machine M, the input x, and the selected oracle $(A_1, \ldots, A_{f(n)})$. In this evaluation queries to A_k are replaced with investigations of the corresponding selection. We notice that M' constructed in this way satisfies (2), alternates $f(n)$ many times, and runs in time $2^{h(n)}$ for some polynomial h.

Only-if part. Let M' be an alternating exponential-time Turing machine with polynomially many alternations. We show how to construct an alternating polynomial-time oracle Turing machine M satisfying (2). W.l.o.g. we find polynomials f and g such that M' runs in time at least n and at most $2^{f(n)} - 2$ and has at most $g(n)$ many alternations.

Let # be a symbol that is not in the alphabet and denote $2^{f(n)} - 1$ by m. Each configuration of M' can be represented as a string

$$\alpha = uqv\# \ldots \#, |\alpha| = m,$$

with the meaning that M' is in state q, has string uv on its tape, and reads the first symbol of string v. The symbol # is only used to pad configurations to the same length. A computation of M' over x may be represented as a sequence of configurations $\alpha_0, \alpha_1, \ldots, \alpha_m$ such that $\alpha_0 = q_0 x\# \ldots \#$ where q_0 is the initial state, $\alpha_m = uqv\# \ldots \#$ where q is some final state, and for $i \leq m-1$ either α_{i+1} is reachable from α_i with one step or $\alpha_i = \alpha_{i+1} = \alpha_m$. Each oracle A_k can encode a computation sequence $\alpha_0^k, \alpha_1^k, \ldots, \alpha_m^k$ with triples $(i, j, \alpha_{i,j}^k)$ where $|i|, |j| \leq f(n)$ and $\alpha_{i,j}^k$ is the jth symbol of configuration α_i^k. Determining whether k, i, j generate a unique $\alpha_{i,j}^k$ can be done with a bounded number of A_k queries since there are only finitely many alphabet and state symbols in M'.

Next we describe the behaviour of the alternating polynomial-time oracle Turing machine M. The idea is to simulate the computation of M' using the

above succinct encoding. M proceeds in $g(n)$ consecutive steps, and below we present step k for $1 \leq k \leq g(n)$ and $Q^k = \exists$. Notice that we use v to indicate the last alternation point of M', i.e., v is a binary string that is initially set to 0 and has always length at most $f(n)$. Notice also that by $\alpha^0_{0,j}$ we refer to the jth symbol of configuration $\alpha_0 = q_0 x\# \dots \#$.

step k:

1. **universally guess** i, j such that $|i|, |j| \leq f(n)$ and $v \leq i$;
   (1a) **if** $\alpha^{k-1}_{v,j} = \alpha^k_{v,j}$ and $\alpha^k_{i,j-1}, \alpha^k_{i,j}, \alpha^k_{i,j+1}, \alpha^k_{i,j+2}$ correctly determine $\alpha^k_{i+1,j}$ **then** proceed to (2);
   (1b) **otherwise** return `false`;
2. **existentially guess** w such that $|w| \leq f(n)$ and $v < w$;
3. **universally guess** i, j such that $|i|, |j| \leq f(n)$ and $v < i < w$;
   (3a) **if** $\alpha^k_{i,j}$ is not a universal state **then** proceed to (4);
   (3b) **otherwise** return `false`;
4. **existentially guess** j such that $|j| \leq f(n)$;
   (4a) **if** $w < m$ and $\alpha^k_{w,j}$ is a universal state **then** set $v \leftarrow w$ and proceed to **step** $k + 1$;
   (4b) **else if** $w = m$ and $\alpha^k_{w,j}$ is an accepting state **then** return `true`;
   (4c) **otherwise** return `false`.

For $1 \leq k \leq g(n)$ and $Q^k = \exists$, step k is described as the dual of the above procedure. Namely, it is obtained by replacing in item (1) universal guessing with existential one, in item (1b) false with true, and in items (3a) and (4a) universal state with existential state. It is now straightforward to check that M runs in polynomial time and satisfies (2). □

Using this theorem we now prove Theorem 7. For the quantification over oracles A_i, we use repetitively $\vee$ and $\sim$.

Theorem 7. SAT(PL[$\sim$]) *is hard for* AEXPTIME(poly).

Proof. Let $A \in$ AEXPTIME(poly). From Theorem 6 we obtain a polynomial f and an alternating oracle Turing machine M with running time bounded by g. By [3], the alternating machine can be replaced by a sequence of word quantifiers over a deterministic Turing machine. (Strictly speaking, [3] speaks only about a bounded number of alternations, but the generalization to the unbounded case is straightforward.) W.l.o.g. we may assume that each configuration of M has at most two configurations reachable in one step. It then follows by Theorem 6 that one can construct a polynomial-time deterministic oracle Turing machine M^* such that $x \in A$ iff

$$Q_1 A_1 \dots Q_{f(n)} A_{f(n)} Q'_1 \boldsymbol{y}_1 \dots Q'_{g(n)} \boldsymbol{y}_{g(n)} \quad (M^* \text{ accepts } (x, \boldsymbol{y}_1, \dots, \boldsymbol{y}_{g(n)}) \text{ with oracle } (A_1, \dots, A_{f(n)})),$$

where $Q_1, \dots, Q_{f(n)}$ and $Q'_1, \dots, Q'_{g(n)}$ are alternating sequences of quantifiers $\exists$ and $\forall$, and each $\boldsymbol{y}_i$ is a $g(n)$-ary sequence of propositional symbols where n is

the length of x. Note that M^* runs in polynomial time also with respect to n. Using this characterization we now show how to reduce in polynomial time any x to a formula φ in PL[$\sim$] such that $x \in A$ iff φ is satisfiable. We construct φ inductively. As a first step, we let

$$\varphi := \max(\boldsymbol{qry}) \wedge p_t \wedge \neg p_f \wedge \varphi_1$$

where

- $\boldsymbol{q}$ and $\boldsymbol{r}$ list propositional symbols that are used for encoding oracles;
- $\boldsymbol{y}$ lists propositional symbols that occur in $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{g(n)}$ and in $\boldsymbol{z}_i$ that are used to simulate configurations of M^* (see phase (3) below);
- p_t and p_f are propositional symbols that do not occur in $\boldsymbol{qry}$.

(1) Quantification over oracles Next we show how to simulate quantification over oracles. W.l.o.g. we may assume that M^* queries binary strings that are of length $h(n)$ for some polynomial h. Let $\boldsymbol{q}$ be a sequence of length $h(n)$ and $\boldsymbol{r}$ a sequence of length $f(n)$. Our intention is that $\boldsymbol{q}$ with r_i encodes the content of the oracle A_i; in fact $\boldsymbol{q}$ and r_i encode the characteristic function of the relation that corresponds to the oracle A_i. For a string of bits $\boldsymbol{b} = b_1 \ldots b_k$ and a sequence $\boldsymbol{s} = (s_1, \ldots, s_k)$ of proposition symbols, we write $\boldsymbol{s} = \boldsymbol{b}$ for $\bigwedge_{i=1}^{k} s_i^{b_i}$, where $s_i^1 := s_i$ and $s_i^0 := \neg s_i$. The idea is that, given a team X over $\boldsymbol{q}\,\boldsymbol{r}$, an oracle A_i, and a binary string $\boldsymbol{a} = a_1 \ldots a_{h(n)}$, the membership of $\boldsymbol{a}$ in A_i is expressed by $X \models \sim\neg(\boldsymbol{q} = \boldsymbol{a} \wedge r_i)$. Note that the latter indicates that there exists $s \in X$ mapping $\boldsymbol{q} \mapsto \boldsymbol{a}$ and $r_i \mapsto 1$. Following this idea we next show how to simulate quantification over oracles A_i. We define φ_i, for $1 \leq i \leq f(n)$, inductively from root to leaves. Depending on whether A_i is existentially or universally quantified, we let

$\exists$: $\varphi_i := \mathrm{dep}(\boldsymbol{q}, r_i) \vee (\mathrm{dep}(\boldsymbol{q}, r_i) \wedge \varphi_{i+1})$,
$\forall$: $\varphi_i := \sim\mathrm{dep}(\boldsymbol{q}, r_i) \otimes (\sim\mathrm{dep}(\boldsymbol{q}, r_i) \owedge \varphi_{i+1})$.

The formula $\varphi_{f(n)+1}$ will be ψ_1 defined in step (2) below. Let us explain the idea behind the definitions of φ_i, first in the case of existential quantification. Assume that X is a team such that

$$X \models \mathrm{dep}(\boldsymbol{q}, r_i) \vee (\mathrm{dep}(\boldsymbol{q}, r_i) \wedge \varphi_{i+1}), \tag{3}$$

and, for $j \geq i$, X is maximal over r_j for all $\boldsymbol{z}_j$, where $\boldsymbol{z}_j$ lists all symbols from the domain of X except r_j. Then by (3) we may choose two subsets $Y, Z \subseteq X$, $Y \cup Z = X$, where $Y \models \mathrm{dep}(\boldsymbol{q}, r_i)$ and $Z \models \mathrm{dep}(\boldsymbol{q}, r_i) \wedge \varphi_{i+1}$. Note that since especially X was maximal over r_i for all $\boldsymbol{q}$, the selection of the partition $Y \cup Z = X$ essentially quantifies over the characteristic functions of the oracle A_i. Moreover, note that, for $j \geq i+1$, Z is maximal over r_j for all $\boldsymbol{z}_j$, where $\boldsymbol{z}_j$ is defined as above.

Universal quantification is simulated analogously. This time we have that

$$X \models \sim\mathrm{dep}(\boldsymbol{q}, r_i) \otimes (\sim\mathrm{dep}(\boldsymbol{q}, r_i) \owedge \varphi_{i+1}), \tag{4}$$

and range over all subsets $Y, Z \subseteq X$ where $Y \cup Z = X$. By (4) for all such Y and Z, we have that if $Y \models \mathrm{dep}(\boldsymbol{q}, r_i)$ and $Z \models \mathrm{dep}(\boldsymbol{q}, r_i)$ then $Z \models \varphi_{i+1}$ (see Section 2.2 for the definition of $\otimes$). Using an analogous argument for Z as in the existential case, we notice that the selection of Z corresponds to universal quantification over characteristic functions of A_i.

(2) Quantification over propositional symbols Next we show how to simulate the quantifier block $Q'_1\boldsymbol{y}_1 \ldots Q'_{g(n)}\boldsymbol{y}_{g(n)}\exists\boldsymbol{z}$ where $\boldsymbol{z}$ lists all propositional symbols that occur in $\boldsymbol{y}$ but not in any $\boldsymbol{y}_i$ (i.e. the remaining symbols that occur when simulating M^*). Assume that this quantifier block is of the form $Q^*_1 y_1 \ldots Q^*_l y_l$, and let $\psi_1 := \varphi_{f(n)+1}$. We define ψ_i again top-down inductively. For $1 \leq i \leq l$, depending on whether Q^*_i is $\exists$ or $\forall$, we let

$\exists$: $\psi_i := \mathrm{dep}(y_i) \vee (\mathrm{dep}(y_i) \wedge \psi_{i+1})$,
$\forall$: $\psi_i := {\sim}\mathrm{dep}(y_i) \otimes ({\sim}\mathrm{dep}(y_i) \varovee \psi_{i+1})$.

Let us explain the idea behind the two definitions of ψ_i. The idea is essentially the same as in the oracle quantification step. First in the case of existential quantification. Assume that we consider a formula ψ_i and a team X where

$$X \models \psi_i, \tag{5}$$

and X is maximal over $y_i \ldots y_l$ for all $\boldsymbol{qr}y_1 \ldots y_{i-1}$. By (5) we may choose two subsets $Y, Z \subseteq X$, $Y \cup Z = X$, where $Y \models \mathrm{dep}(y_i)$ and $Z \models \mathrm{dep}(y_i) \wedge \psi_{i+1}$. There are now two options: either we choose $Z = \{s \in X \mid s(y_i) = 0\}$ or $Z = \{s \in X \mid s(y_i) = 1\}$. Since X is maximal over $y_i \ldots y_l$ for all $\boldsymbol{qr}y_1 \ldots y_{i-1}$, we obtain that $Z \upharpoonright \boldsymbol{qr} = X \upharpoonright \boldsymbol{qr}$ and Z is maximal over $y_{i+1} \ldots y_l$ for all $\boldsymbol{qr}y_1 \ldots y_i$. Hence no information about oracles is lost in this quantifier step.

The case of universal quantification is again analogous to the oracle case. Hence we obtain that (5) holds iff both $\{s \in X \mid s(y_i) = 0\}$ and $\{s \in X \mid s(y_i) = 1\}$ satisfy ψ_{i+1}.

(3) Simulation of computations Next we define $\psi_{g(n)+1}$ that simulates the polynomial-time deterministic oracle Turing machine M^*. Note that this formula is evaluated over a subteam X such that $X \models \mathrm{dep}(y_i)$, for each $y_i \in \boldsymbol{y}$, and $\boldsymbol{a} \in A_i$ iff $X \models {\sim}\neg(\boldsymbol{q} = \boldsymbol{a} \wedge r_i)$. Using this it is now straightforward to construct a propositional formula θ such that $\exists\boldsymbol{c}(X[\boldsymbol{b}_i/\boldsymbol{y}_i][\boldsymbol{c}/\boldsymbol{z}] \models \theta)$ if and only if M^* accepts $(x, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{g(n)})$ with oracle $(A_1, \ldots, A_{f(n)})$. Here $X[\boldsymbol{a}/\boldsymbol{x}]$ denotes the team $\{s(\boldsymbol{a}/\boldsymbol{x}) : s \in X\}$ where $s(\boldsymbol{a}/\boldsymbol{x})$ agrees with s everywhere except that it maps x pointwise to $\boldsymbol{a}$. Each configuration of M^* can be encoded with a binary sequence $\boldsymbol{z}_i$ of length $O(t(n))$ where t is a polynomial bounding the running time of M^*. Then it suffices to define ψ_{l+1} as a conjunction of formulae $\theta_{\mathrm{start}}(\boldsymbol{z}_0), \theta_{\mathrm{move}}(\boldsymbol{z}_i, \boldsymbol{z}_{i+1}), \theta_{\mathrm{final}}(\boldsymbol{z}_{t(n)})$ describing that $\boldsymbol{z}_0$ corresponds to the initial configuration, $\boldsymbol{z}_i$ determines $\boldsymbol{z}_{i+1}$, and $\boldsymbol{z}_{t(n)}$ is in accepting state. Note that the formulae $\theta_{\mathrm{start}}(\boldsymbol{z}_0)$, $\theta_{\mathrm{move}}(\boldsymbol{z}_i, \boldsymbol{z}_{i+1})$, and $\theta_{\mathrm{final}}(\boldsymbol{z}_{t(n)})$ can be written exactly as in the classical setting, except that all disjunctions $\vee$ are replaced by the intuitionistic disjunction $\varovee$.

Finally note that, by Proposition 4, all occurrences of dependence atoms, the shorthand $\max(\cdot)$, and the connectives $\otimes$ and $\otimes$ can be eliminated from the above formulae by a polynomial overhead. Thus the constructed formula φ is a $\mathrm{PL}[\sim]$-formula as required.

By Proposition 6, and Theorems 5 and 7 we now obtain the following.

Theorem 8. *Satisfiability and validity of* $\mathrm{PL}[\perp_c, \subseteq, \sim]$ *and* $\mathrm{PL}[\sim]$ *are complete for* $\mathsf{AEXPTIME}(\mathsf{poly})$.

The following corollary now follows by a direct generalisation of Theorem 5.

Corollary 1. *Let $\mathcal{C}$ be a finite collection of dependency atoms with polynomial-time checkable semantics. Satisfiability and validity of* $\mathrm{PL}[\mathcal{C}, \sim]$ *is complete for* $\mathsf{AEXPTIME}(\mathsf{poly})$.

4 Complexity of Model Checking

In this section we consider the model checking problems of our logics. We first focus on logics without the classical negation.

Theorem 9. $\mathsf{MC}(\mathrm{PL}[\perp_c])$ *is complete for* NP.

Proof. The upper bound follows since the model checking problem for modal independence logic is NP-complete [13]. Since dependence atoms can be expressed efficiently by independence atoms (see the proof of Theorem 2), the lower bound follows from the NP-completeness of $\mathsf{MC}(\mathrm{PD})$ (see Table 2).

The following unpublished result was shown by Hella et al.

Theorem 10 ([11]). $\mathsf{MC}(\mathrm{PL}[\subseteq])$ *is* P*-complete.*

The following result can also be found in the PhD thesis of Müller [19].

Theorem 11. $\mathsf{MC}(\mathrm{PL}[\sim])$ *is complete for* PSPACE.

Proof. For the upper bound note that Algorithm 1 decides the problem in APTIME which is exactly PSPACE [3]. For the lower bound, we reduce from TQBF which is known to be PSPACE-complete. Let $Q_1x_1 \dots Q_nx_n\theta$ be a quantified boolean formula. Let $\boldsymbol{r}$ be a sequence of propositional symbols of length $\log(n)+1$, and let $T := \{s_1, \dots, s_n\}$ be a team where $s_i(\boldsymbol{r})$ writes i in binary. We define inductively a formula $\varphi \in \mathrm{PL}[\sim]$ such that

$$Q_1x_1 \dots Q_nx_n\theta \text{ is true iff } T \models \varphi. \tag{6}$$

Let $\varphi := \varphi_1$, and for $1 \le i \le n$, depending on whether x_i is existentially or universally quantified we let

$\exists$: $\varphi_i := \boldsymbol{r} = \mathrm{bin}(i) \vee \varphi_{i+1}$,
$\forall$: $\varphi_i := {\sim}\boldsymbol{r} = \mathrm{bin}(i) \otimes \varphi_{i+1}$.

Finally, we let φ_{n+1} denote the formula obtained from θ by first substituting each $\neg x_i$ by $\neg \boldsymbol{r} = \text{bin}(i)$ and then x_i by $\sim\neg \boldsymbol{r} = \text{bin}(i)$, for each i. Note that the meaning $\neg \boldsymbol{r} = \text{bin}(i)$ is that the assignment s_i is not in the team, whereas $\sim\neg \boldsymbol{r} = \text{bin}(i)$ states that s_i is in the team. It is now straightforward to establish that (6) holds. Also T and φ can be constructed in polynomial time, and hence we obtain the result. □

Since Algorithm 1 can also be applied to independence and inclusion atoms, we obtain the following corollary.

Corollary 2. $\mathsf{MC}(\mathrm{PL}[\perp_\mathrm{c}, \subseteq, \sim])$ *and* $\mathsf{MC}(\mathrm{PL}[\mathcal{C}, \sim])$*, where* $\mathcal{C}$ *is a finite collection of polynomial time computable dependency atoms, are complete for* PSPACE*.*

5 Conclusion

In this article we have initiated a systematic study of the complexity theoretic properties of team based propositional logics. Regarding the logics considered in this paper, an interesting open question is to determine the exact complexity of $\mathsf{VAL}(\mathrm{PL}[\perp_\mathrm{c}])$ for which membership in $\mathsf{coNEXPTIME}^{\mathsf{NP}}$ was shown in this paper. Propositional team semantics is a very rich framework in which many interesting connectives and operators can be studied such as the intuitionistic implication $\multimap$ applied in the area of inquisitive semantics. It is an interesting question to extend this study to cover a more wide range of team based logics.

References

1. Samson Abramsky and Jouko Väänänen. From IF to BI. *Synthese*, 167:207–230, 2009. 10.1007/s11229-008-9415-6.
2. Sam Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 123–131, New York, NY, USA, 1987. ACM.
3. Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, January 1981.
4. Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.
5. Arnaud Durand, Juha Kontinen, and Heribert Vollmer. Expressivity and complexity of dependence logic. In Samson Abramsky, Juha Kontinen, Jouko Väänänen, and Heribert Vollmer, editors, *Dependence Logic: Theory and Applications*, pages 5–32. Springer International Publishing, 2016.
6. Johannes Ebbing and Peter Lohmann. Complexity of model checking for modal dependence logic. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 226–237. Springer Berlin / Heidelberg, 2012.
7. Pietro Galliani. Inclusion and exclusion dependencies in team semantics: On some logics of imperfect information. *Annals of Pure and Applied Logic*, 163(1):68 – 84, 2012.

8. Erich Grädel and Jouko Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013.
9. Miika Hannula and Juha Kontinen. A finite axiomatization of conditional independence and inclusion dependencies. *Inf. Comput.*, 249:121–137, 2016.
10. Miika Hannula, Juha Kontinen, Martin Lück, and Jonni Virtema. On quantified propositional logics and the exponential time hierarchy. In Domenico Cantone and Giorgio Delzanno, editors, Proceedings of the Seventh International Symposium on *Games, Automata, Logics and Formal Verification,* Catania, Italy, 14-16 September 2016, volume 226 of *Electronic Proceedings in Theoretical Computer Science*, pages 198–212. Open Publishing Association, 2016.
11. Lauri Hella, Antti Kuusisto, Arne Meier, and Jonni Virtema. Model checking and validity in propositional and modal inclusion logics. *CoRR*, abs/1609.06951, 2016.
12. Lauri Hella, Antti Kuusisto, Arne Meier, and Heribert Vollmer. Modal inclusion logic: Being lax is simpler than being strict. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *MFCS (1)*, volume 9234 of *Lecture Notes in Computer Science*, pages 281–292. Springer, 2015.
13. Juha Kontinen, Julian-Steffen Müller, Henning Schnoor, and Heribert Vollmer. A Van Benthem Theorem for Modal Team Semantics. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, pages 277–291, 2015.
14. Juha Kontinen and Ville Nurmi. Team logic and second-order logic. *Fundam. Inform.*, 106(2-4):259–272, 2011.
15. Leonid A. Levin. Universal search problems. *Problems of Information Transmission*, 9(3), 1973.
16. Peter Lohmann and Heribert Vollmer. Complexity results for modal dependence logic. *Studia Logica*, 101(2):343–366, 2013.
17. Martin Lück. Axiomatizations for Propositional and Modal Team Logic. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:18, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
18. Martin Lück. Complete problems of propositional logic for the exponential hierarchy. *CoRR*, abs/1602.03050, 2016.
19. Julian-Steffen Müller. Satisfiability and model checking in team based logics. *PhD Thesis, Leibniz Universität Hannover, Cuvillier Verlag Göttingen*, 2014.
20. Pekka Orponen. Complexity classes of alternating machines with oracles. In *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*, pages 573–584, 1983.
21. Katsuhiko Sano and Jonni Virtema. Axiomatizing Propositional Dependence Logics. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 292–307, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
22. Jouko Väänänen. *Dependence Logic*. Cambridge University Press, 2007.
23. Jonni Virtema. Complexity of validity for propositional dependence logics. In Adriano Peron and Carla Piazza, editors, *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, volume 161 of *EPTCS*, pages 18–31, 2014.
24. Fan Yang. *On Extensions and Variants of Dependence Logic*. PhD thesis, University of Helsinki, 2014.
25. Fan Yang and Jouko Väänänen. Propositional logics of dependence. *Ann. Pure Appl. Logic*, 167(7):557–589, 2016.

Algorithm 1 APTIME algorithm for MC(PL[$\perp_c, \subseteq, \sim$])

function $\textsc{mc}(T, \varphi, I)$
  if $\varphi = \psi_1 \wedge \psi_2$ **then**
    if $I = 1$ **then**
      universally choose $i \in \{1, 2\}$
      return $\textsc{mc}(T, \psi_i, I)$
    else if $I = 0$ **then**
      existentially choose $i \in \{1, 2\}$
      return $\textsc{mc}(T, \psi_i, I)$
  else if $\varphi = \psi_1 \vee \psi_2$ **then**
    if $I = 1$ **then**
      existentially choose $T_1 \cup T_2 = T$
      universally choose $i \in \{1, 2\}$
      return $\textsc{mc}(T_i, \psi_i, I)$
    else if $I = 0$ **then**
      universally choose $T_1 \cup T_2 = T$
      existentially choose $i \in \{1, 2\}$
      return $\textsc{mc}(T_i, \psi_i, I)$
  else if $\varphi = \sim\psi$ **then**
    if $I = 1$ **then**
      return $\textsc{mc}(T, \psi, 0)$
    else if $I = 0$ **then**
      return $\textsc{mc}(T, \psi, 1)$
  else if $\varphi = p$ ($\varphi = \neg p$) **then**
    $1 \leftarrow x$
    for $s \in T$ **do**
      if $s(p) = 0$ ($s(p) = 1$) **then**
        $0 \leftarrow x$
    if $x = I = 1$ **or** $x = I = 0$ **then**
      return true
    else
      return false
  else if $\varphi = \boldsymbol{p} \subseteq \boldsymbol{q}$ **then**
    $1 \leftarrow x$
    for $s \in T$ **do**
      $0 \leftarrow y$
      for $s' \in T$ **do**
        if $s(\boldsymbol{p}) = s'(\boldsymbol{q})$ **then**
          $1 \leftarrow y$
      if $y = 0$ **then**
        $0 \leftarrow x$
    if $x = I = 1$ **or** $x = I = 0$ **then**
      return true
    else
      return false
  else if $\varphi = \boldsymbol{q} \perp_{\boldsymbol{p}} \boldsymbol{r}$ **then**
    $1 \leftarrow x$
    for $s, s' \in T$ **with** $s(\boldsymbol{p}) = s'(\boldsymbol{p})$ **do**
      $0 \leftarrow y$
      for $s'' \in T$ **do**
        if $s(\boldsymbol{p}) = s''(\boldsymbol{p}), s(\boldsymbol{q}) = s''(\boldsymbol{q}), s'(\boldsymbol{r}) = s''(\boldsymbol{r})$ **then**
          $1 \leftarrow y$
      if $y = 0$ **then**
        $0 \leftarrow x$
    if $x = I = 1$ **or** $x = I = 0$ **then**
      return true
    else
      return false